Collaborative Management of Tasks Tool

Xiaoxu Tan, Xinming Liao, Sarbojit Bhattacharjee, Felicita Florence Clement, Steffi Evangeline Banans

I. Introduction

The main goal of the project is to develop a collaborative task management tool with versatile functionalities. The initial phase involves designing a login interface with a signup option and a checkbox for users to register their access. Subsequently, our focus will shift towards constructing a dashboard equipped with features like task addition, project details, deadline setting, progress tracking, and task assignment.

II. Technology & Architecture

UI Design

In the initial phase of our project, we focused on designing the user interface. Employing wireframe techniques, we drew inspiration from monday.com as our foundational template, aiming to create an intuitive and user-friendly design.

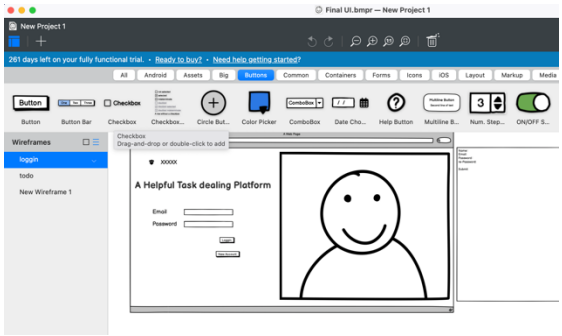

Figure 1: UI design for login

Backend

We setup server after the UI confirmed:

- Core: Node.js with Express for RESTful API design.
- Database: Mongoose for MongoDB interactions.
- Auth: jsonwebtoken for handling authentication.
- Security: bcryptjs for encrypting passwords.
- Cookies: cookie-parser to manage cookies.
- Logging: morgan for tracking HTTP requests.
- Debugging: debug module for identifying issues.
- Development Tool: nodemon for auto-restarting the server on code changes.
- Entry Point: The app.js file, with start script invoking ./bin/www via nodemon.

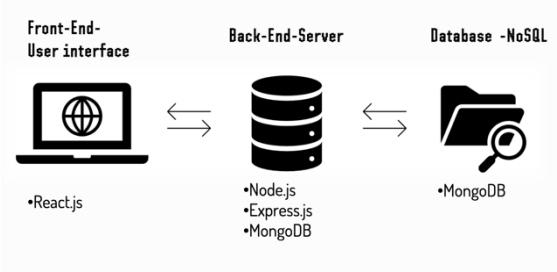

Figure 2: the MERN stack

Our backend is designed with a modular code structure, which makes maintenance and testing easier. We achieved this by encapsulating Mongoose's functions for querying the database in a db.js file and exporting them to the app.js file. The Express uses these functions to handle routing and define endpoints for API requests, such as retrieving task information or updating user details.

Organizing the code in this way allows us to test functions at different levels, such as the base function level and the API level for Tasks management. We use Postman for backend testing, which allows us to simulate database interactions and simulate scenarios to verify that functions actually behave as expected. Using Postman also allows us to test

functionality before the database setup is complete.

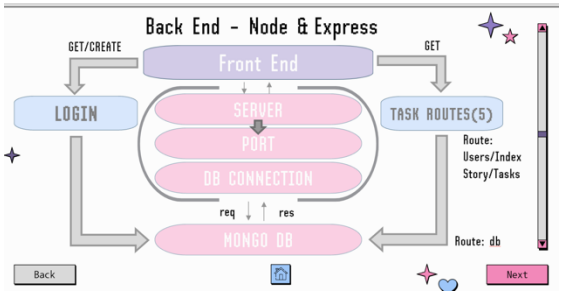

Figure 3: the backend flow

After we confirm that the code functions correctly, we proceed with setting up the database using MongoDB.

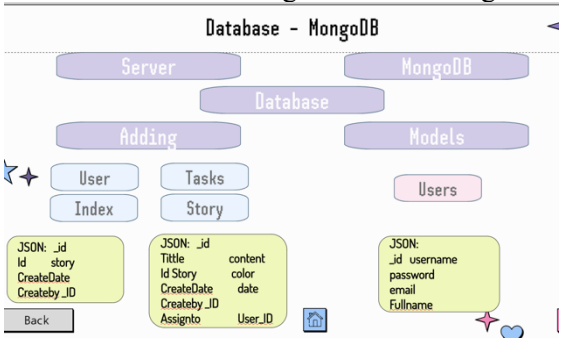

Figure 4: Database

The database structure is as shown in the figure. User information and task details are stored in JSON format, ensuring the readability and accessibility of the data. The user model contains information such as username, password, email, and full name, while the task model includes fields such as title, content, status, and creation date.

We implemented sufficient checks on the front end to ensure that sufficient resources are available when a user books a task. This includes checking the number of tasks a user has booked against the number of tasks available remaining in the database to avoid over-allocation of resources.

Frontend

Our application uses React.js, a widely used framework that simplifies the process of creating and maintaining web applications. Its component reusability allows developers to reduce redundancy and enhance consistency throughout the application. React is great for building single-page applications that are responsive and mobile-friendly. Our application has dynamic styling capabilities, using JSX to allow developers to dynamically change the look and feel of the application. We also took advantage of React's state management hooks, such as useState and useEffect, to effectively manage component state and side effects.

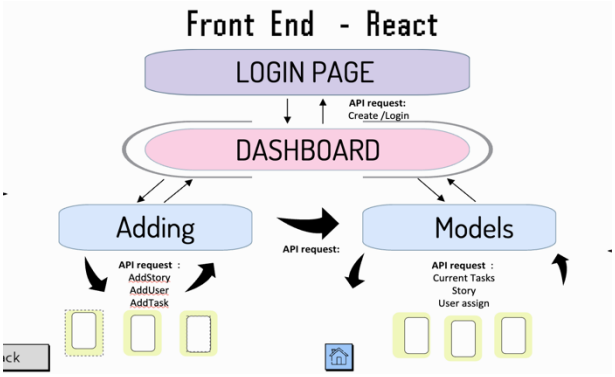

Figure 5: React structure

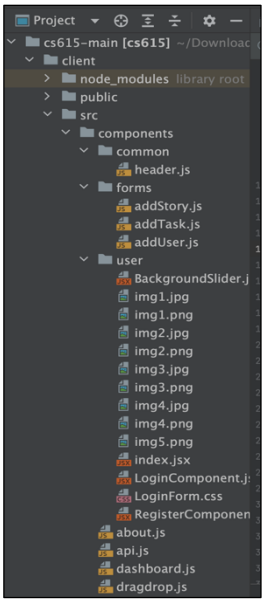

figure 5: modular code

Our code structure is well organized, with the code for different features such as routing, user authentication, card management, and main dashboard neatly organized in different folders for easy updates and maintenance. Our React application structure is designed to provide users with a consistent operating experience no matter where they are in the application. Using React router to manage in-app navigation, we map specific URLs to corresponding components to achieve dynamic route matching.

The front end uses react-scripts, our start script quickly starts the development environment, and the build script provides optimized code for production deployment. The test script helps us maintain high quality code, while the eject script gives advanced users the ability to customize build configurations. start script: Use

react-scripts to quickly start the development server of React applications, making front-end development and testing faster. These scripts provide a simple interface through react-scripts to manage a variety of development tasks, without requiring in-depth knowledge of the underlying build and test configurations, providing convenience to developers. In addition, we also set up a proxy to allow the front-end development server to proxy API requests to the back-end server to facilitate front-end and back-end separated development.

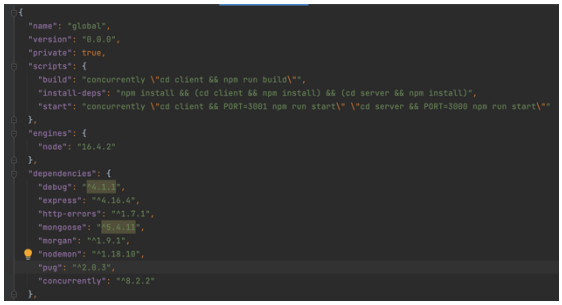

Figure6: Global Script

We specially designed a global script. Our global build script effectively shortens the overall build time by concurrently building the client and server in parallel. The install-deps script simplifies dependency management, installing required dependencies for the entire project with a single command. The global start script starts the front and back ends at the same time, allowing us to work in a unified development process.

III. Testing and Version Control

Backend

We use setuptest.js; to test the software back-end. setuptest.js files are typically used to set up the test environment, configure the test framework, and perform the necessary initialisation operations. We imported the required test libraries and modules in the setuptest.js file, configured the test environment (e.g., connect to the test database), and performed any necessary global settings.

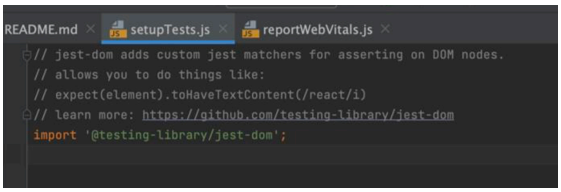

Figure7: setuptest.js

Many companies use their own scripts to test software frameworks etc., so we looked for open- source scripts from GitHub and tried to solve the testing using the company's approach;

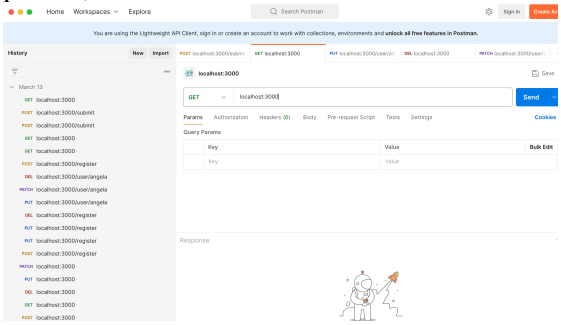

Figure8: Postman testing

The same features in Postman are required to pay, but we also did not give up using Postman through the independent post and get, which helped our less capable group members better understand the API operation in the code.

Frontend

We performed the following tests:
Layout and rendering: Operating systems and browsers differed, and we performed groupings to verify that UI elements were correctly positioned, aligned, and rendered on different devices, screen sizes, and resolutions;

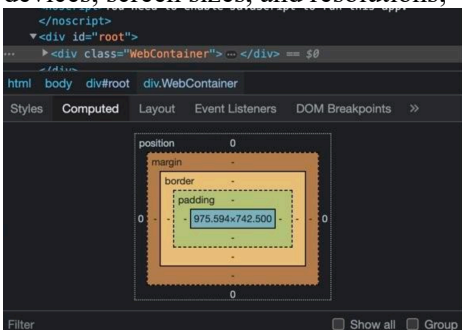

Figure8: Frame testing

Forms and inputs: the input functionality of our login screen, grouped for testing and feedback.

Buttons and Actions: Validate the functionality of buttons, including click actions, enable/disable states, and the proper execution of associated actions.

Error handling: testing how the application handles and displays errors, alerts and validation messages, ensuring that they are user-friendly and informative.

Version Control—Github

During the development of our projects, we strictly followed the best practices of using GitHub for version control. We ensure that each team member works on their own branch, which helps isolate development and prevent unstable code from affecting the main branch (main)

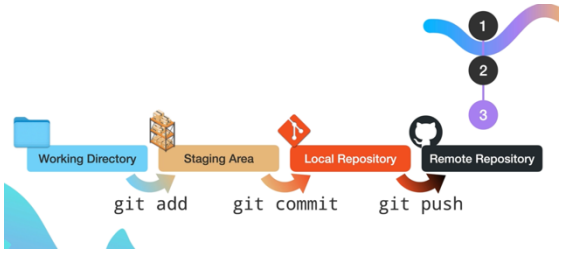

Figure8: GitHub version control

Development process and version control strategy:

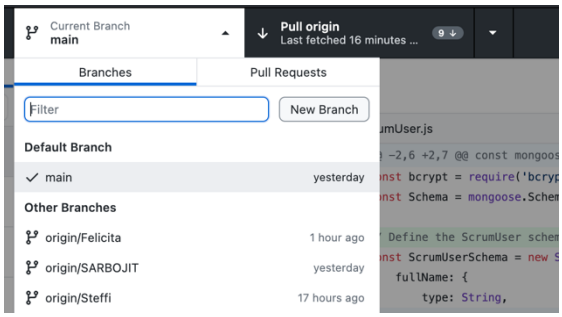

Figure9: Branch strategy

Branch strategy: Each member develops on our own branch. These branches are usually named after functions or developers. Using branches can ensure the stability of the main branch. All new features and fixes are completed and tested on their respective branches. Only then can it be merged back into the main branch.

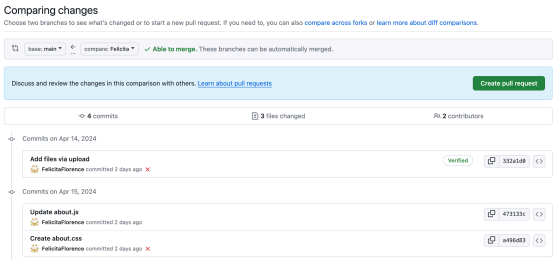

Figure10: review and merge

Code review and merge: We use the Pull Request (PR) mechanism to review code. When a feature is developed, the developer will initiate a PR to the main branch.

Other members of the team will review the PR, make comments or suggestions, and make modifications if necessary to ensure code quality and consistency.

Only when all core team members agree can it be merged into the main branch. This process enhances code quality and team collaboration.

Canonical commit message: We encourage team members to use clear and descriptive commit messages, which makes history easier to trace and future code maintenance.

Through this strict version control process, our projects maintain a high level of code quality and maintainability. At the same time, this also promotes collaboration and communication among team members, and everyone can maintain a clear understanding of the progress and changes of the project.

Conclusion:
In summary, our team effectively crafted and implemented a collaborative task management. tool, enabling users to establish accounts and navigate an interactive dashboard for task. oversight and administration. The project successfully met its key objectives by prioritizing. user-friendly functionality and fostering individual team member growth. Each member gained insights into the significance of their contributions, honed existing skills, acquired new ones, and navigated challenges, thus preparing them for future projects in the competitive labour market

[Appendix: Weekly Meeting and process]

UI Design Progress

Location：Lab

Date: March 16, 2024

In this initial phase, our team focused on a user-friendly UI, drawing from monday.com's intuitive layout and selecting 10 templates from Dribbble and Behance to inspire our design. Intense online discussions helped refine our choices, leading to a streamlined UI confirmed through team feedback and an email poll.

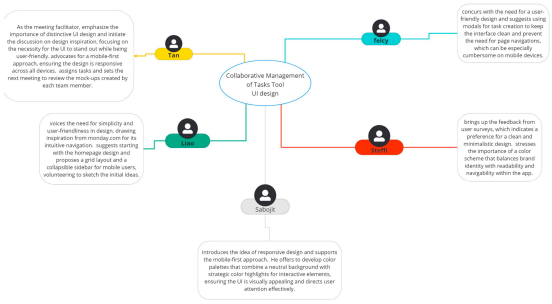

Figure 1: UI design on Miro

Backend Development Progress

Location：Lab

Date: March 25, 2024

We established our server foundation using Node.js with Express and integrated Mongoose with MongoDB. Security features included jsonwebtoken for authentication and bcryptjs for password encryption. Our backend setup also featured cookie-parser and morgan for session and request logging, respectively, with nodemon for development workflow automation, complemented by Postman for API testing pre-database setup.

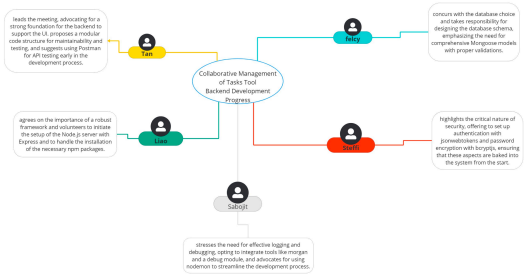

Figure 2: Front-end on Miro

Front-end Production

Location：Lab

Date: April 4, 2024

The front-end phase involved implementing the main pages with HTML, CSS, and JavaScript, emphasizing animations and layout with React.js for an engaging user interface. We capitalized on React's state management hooks and modular code structure, facilitating development with a global script that streamlined concurrent client-server builds.

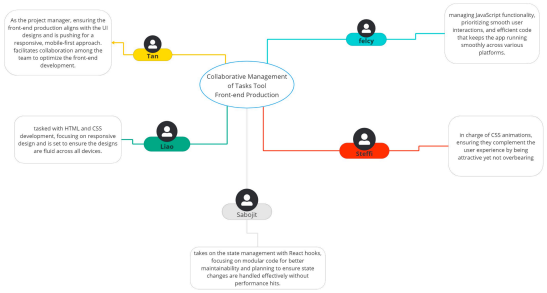

Figure 2: Back-end on Miro

Testing

Location：Teams

Date: April 11, 2024

Utilizing react-scripts, we handled server starts, production builds, and quality assurance testing. Our proxy setup enabled efficient front-end and backend API communication, ensuring a seamless development cycle and focused debugging efforts.Code debugging is a critical part of ensuring software quality and stability.

Fixing bugs and monitoring code changes using nodemon: In this phase, we focused on fixing the bugs and code defects found. Tools such as Visual Studio Code and ESLint were used to identify issues, and the nodemon tool was utilized to monitor code changes and update the server in real-time to quickly test code changes.

Testing the API using Postman: In addition, the Postman tool was used to continue testing the functionality of all API endpoints to ensure that the API was reliable and performed as expected.

UX Testing is the final phase of the project.

Commit, Merge and review

Location：Teams

Date: April 11, 2024

We discussed strategies for optimizing our commit, merge, and review processes to enhance collaboration and maintain high standards of code quality in the TaskManagement project. Emphasis was placed on the need for clear and descriptive commit messages, stringent peer reviews before merges, and the use of automated testing tools to detect issues early. The team agreed to adopt a more structured workflow to prevent bottlenecks and ensure smooth integration of new code contributions.

Group Discussion: Based on the feedback, team members held a group discussion to analyse the user experience and the ease of use of the software and identified a number of areas for improvement.

Code and Interface Adjustment: Based on the results of the discussion, we adjusted and optimized the details of the code and interface design to improve the overall user satisfaction and operational efficiency of the software.